باسمه‌تعالی

پروژه DSD فاز یک و دو

دکتر فصحتی

نیم‌سال دوم سال تحصیلی ۱۴۰۱-۱۴۰۰

اعضای گروه: محمدپرهام باطنی، امیرمحمد ایزدی، پویا اسمعیل آخوندی

شماره‌های دانشجویی:

محمدپرهام باطنی: ۹۹۱۰۵۲۹۴

امیرمحمد ایزدی: ۹۹۱۰۵۲۸۳

پویا اسماعیلی آخوندی: ۹۹۱۰۹۳۰۳

ایمیل‌ها:

mp.bateni@gmail.com

amirmmdizady@gmail.com

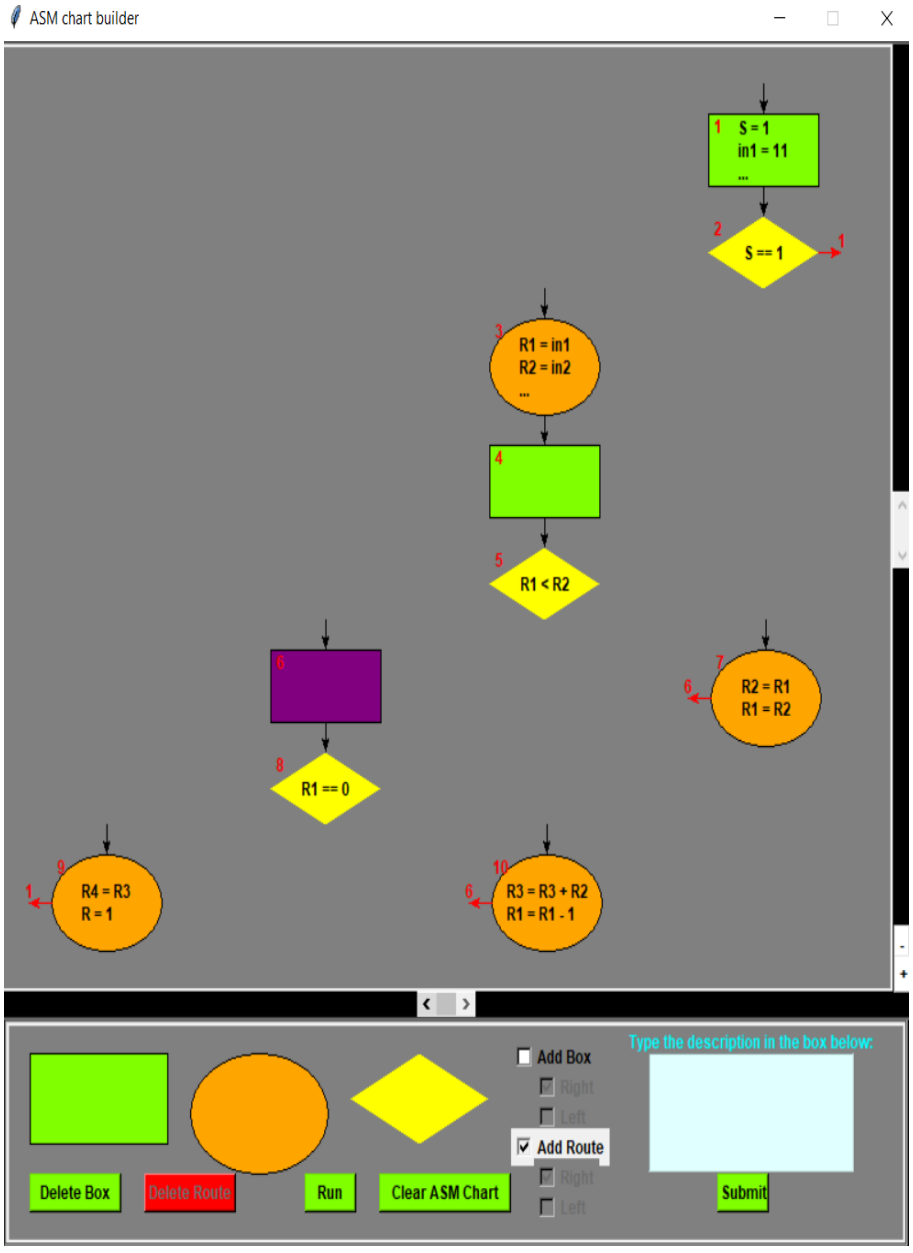
pooyaesmakh.81@yahoo.com

توضیحات مربوط به کار با ابزار گرافیکی در وویس توضیحات کلی پروژه گفته شده است!

فاز یک: شبیه‌سازی ASM chart

تست یک: تست مدار ضرب کننده گفته شده در کلاس

نمای ASM Chart ورودی به برنامه :



مطابق ASM داده شده خروجی ما شبیه سازی شده آن است، یعنی در هر کلاک مقادیر متغیر ها را در فایل قرار می دهیم، تصویر چند خط اول و آخر تست 1 را در ادامه می توانید ببینید (چون تعداد خط ها زیاد بود برای جلوگیری

از شلوغ شدن همه خطوط را نیاوردیم ولی خروجی تست 1 به صورت کامل در فایل variables_table_Test1.txt قرار دارد میتوانید مشاهده کنید)

عکس های خروجی شبیه‌سازی :

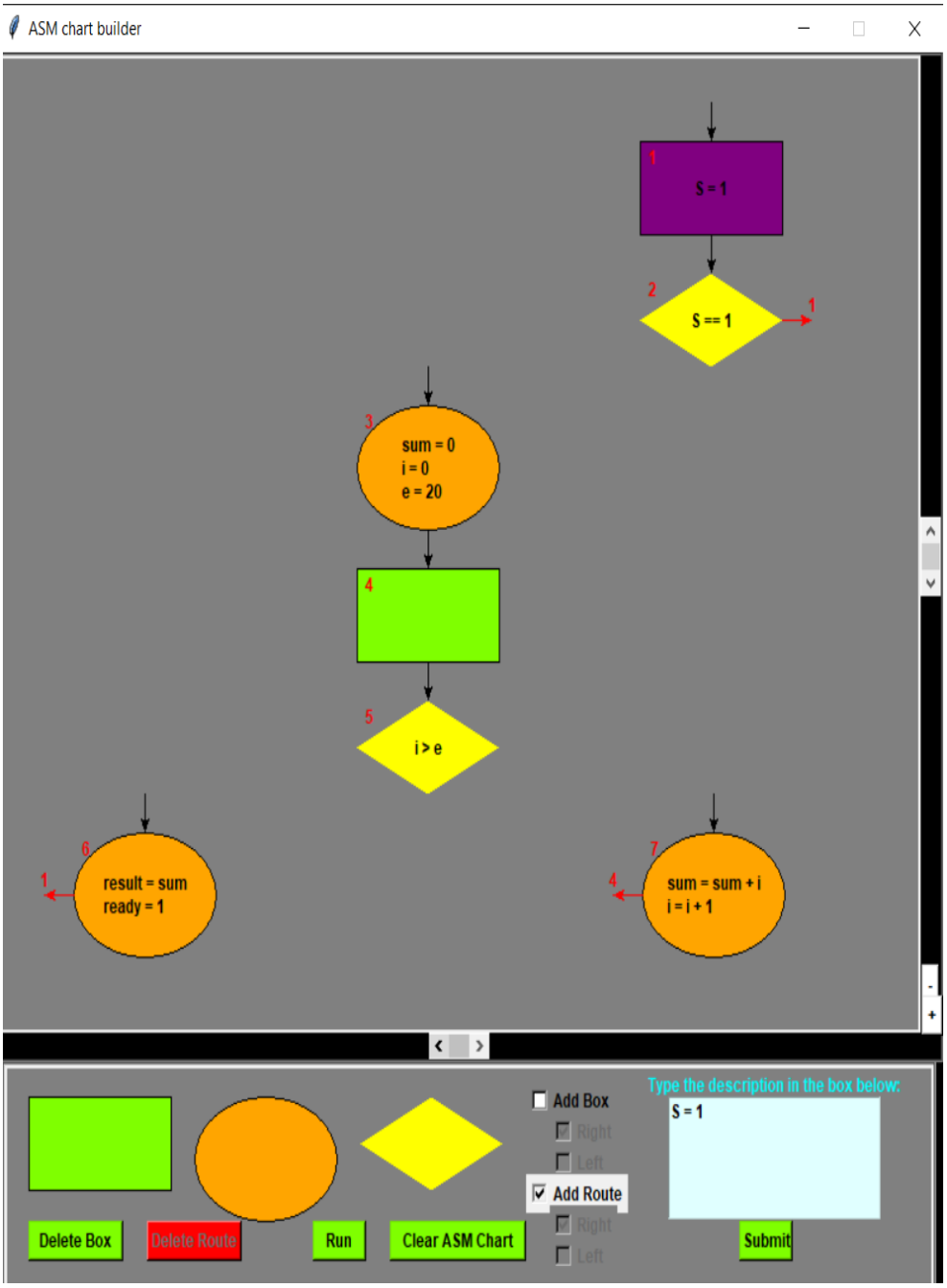
```
Cycle: 1
ASM Block: 1
Variable        Value
S       1
in1     11
in2     13
************************
Cycle: 2
ASM Block: 4
Variable        Value
S       1
in1     11
in2     13
R1      11
R2      13
R3      0
R       0
************************
Cycle: 3
ASM Block: 6
Variable        Value
S       1
in1     11
in2     13
R1      11
R2      13
R3      0
R       0
************************
Cycle: 4
ASM Block: 6
Variable        Value
S       1
in1     11
in2     13
R1      10
R2      13
R3      13
R       0
************************
Cycle: 5
ASM Block: 6
Variable        Value
S       1
in1     11
```

```
Cycle: 12
ASM Block: 6
Variable        Value
S       1
in1     11
in2     13
R1      2
R2      13
R3      117
R       0
************************
Cycle: 13
ASM Block: 6
Variable        Value
S       1
in1     11
in2     13
R1      1
R2      13
R3      130
R       0
************************
Cycle: 14
ASM Block: 6
Variable        Value
S       1
in1     11
in2     13
R1      0
R2      13
R3      143
R       0
************************
Cycle: 15
ASM Block: 1
Variable        Value
S       1
in1     11
in2     13
R1      0
R2      13
R3      143
R       1
R4      143
************************
```

همانطور که در تصاویر بالا مشخص است خروجی R4 = 143 و R = 1 شده است که مشخصاً خروجی مطلوب ما است.

تست دو: تست جمع اعداد 1 تا n (برای این ورودی 1 تا 20)

نمای ASM Chart ورودی به برنامه :



مطابق ASM داده شده خروجی ما شبیه سازی شده آن است، یعنی در هر کلاک مقادیر متغیر ها را در فایل قرار می دهیم، تصویر چند خط اول و آخر تست 2 را در ادامه می توانید ببینید (چون تعداد خط ها زیاد بود برای جلوگیری

از شلوغ شدن همه خطوط را نیاوردیم ولی خروجی تست 2 به صورت کامل در فایل variables_table_Test2.txt قرار دارد میتوانید مشاهده کنید)

عکس های خروجی شبیه‌سازی :

```
Cycle: 1
ASM Block: 1
Variable        Value
S       1
************************
Cycle: 2
ASM Block: 4
Variable        Value
S       1
sum     0
i       0
e       20
************************
Cycle: 3
ASM Block: 4
Variable        Value
S       1
sum     0
i       1
e       20
************************
Cycle: 4
ASM Block: 4
Variable        Value
S       1
sum     1
i       2
e       20
************************
Cycle: 5
ASM Block: 4
Variable        Value
S       1
sum     3
i       3
e       20
************************
Cycle: 6
ASM Block: 4
Variable        Value
S       1
sum     6
i       4
e       20
************************
```

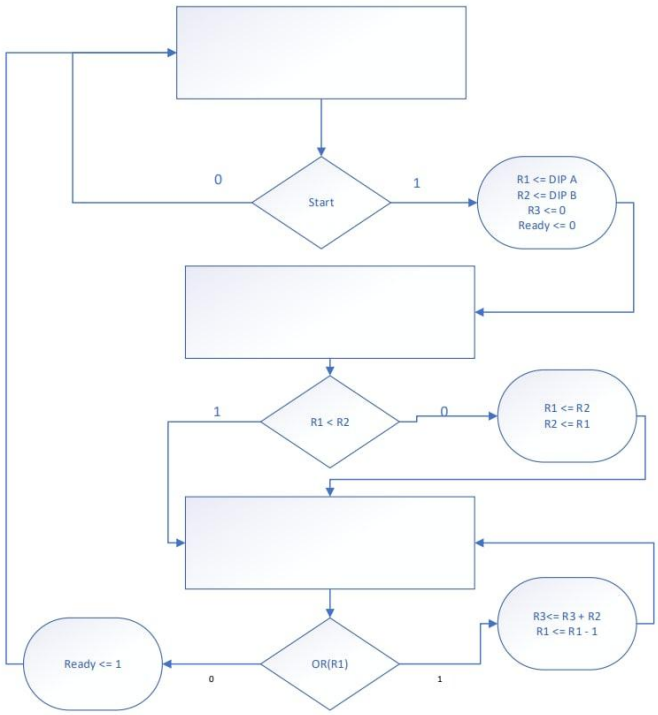
```
Cycle: 20
ASM Block: 4
Variable        Value
S       1
sum     153
i       18
e       20
************************
Cycle: 21
ASM Block: 4
Variable        Value
S       1
sum     171
i       19
e       20
************************
Cycle: 22
ASM Block: 4
Variable        Value
S       1
sum     190
i       20
e       20
************************
Cycle: 23
ASM Block: 4
Variable        Value
S       1
sum     210
i       21
e       20
************************
Cycle: 24
ASM Block: 1
Variable        Value
S       1
sum     210
i       21
e       20
result  210
ready   1
************************
```

همانطور که در تصاویر بالا مشخص است خروجی result = 210 و ready = 1 شده است که مشخصاً خروجی مطلوب ما است (جمع اعداد 1 تا 20، 210 می‌شود).

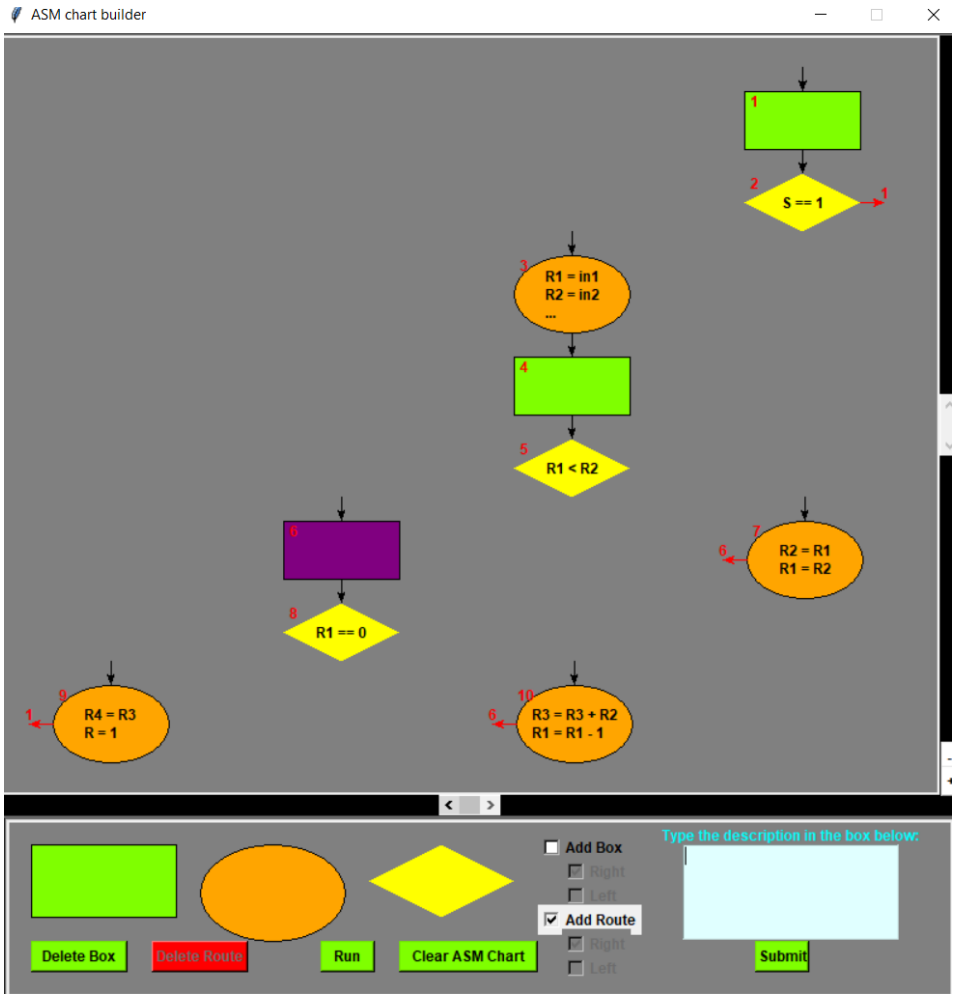
فاز دو: تبدیل ASM chart به کد وریلاگ

تست یک: تست مدار ضرب‌کننده گفته شده در کلاس درس

شماتیک ASM chart:



نمای ASM chart در برنامه:



ماژول‌های تولید شده:

control_unit.v:

```
module control_unit(Enable3, Enable7, Enable9, Enable10, output2, output5, output8, clk, rst);
input [31:0] output2, output5, output8;
output reg Enable3, Enable7, Enable9, Enable10;
input clk, rst;
integer p_state, n_state;

always @ (posedge clk)
begin
if (rst == 1'b0) begin p_state = 1; end
else begin p_state = n_state; end
end

always @(p_state, output2, output5, output8)
begin
begin
Enable3 = 0;
Enable7 = 0;
Enable9 = 0;
Enable10 = 0;
end
case(p_state)
1:
begin
if (output2 == 1) begin
begin Enable3 = 1; end
begin n_state = 4; end
end else begin
begin n_state = 1; end
end
end
4:
begin
if (output5 == 1) begin
begin n_state = 6; end
end else begin
begin Enable7 = 1; end
begin n_state = 6; end
end
end
6:
begin
if (output8 == 1) begin
begin Enable9 = 1; end
begin n_state = 1; end
end else begin
begin Enable10 = 1; end
begin n_state = 6; end
end
end
endcase
end
endmodule
```

data_path.v:

```
module data_path(R3, in1, R4, in2, R, S, output2, Enable3, output5, Enable7, output8, Enable9, Enable10, clk);
input Enable3, Enable7, Enable9, Enable10;
output [31:0] output2, output5, output8;
input clk;
input [31:0] in1;
input [31:0] in2;
input [31:0] S;
output [31:0] R3;
output [31:0] R4;
output [31:0] R;
wire [31:0] R1;
wire [31:0] R2;
module2 create_module2(S, output2);
module5 create_module5(R1, R2, output5);
module8 create_module8(R1, output8);
total_module total_module1(Enable3, Enable7, Enable9, Enable10, R3, in1, R4, R1, in2, R, R2, clk);
endmodule
```

main_program.v:

```
module main_program(in1, in2, S, R3, R4, R, clk, rst);
wire Enable3, Enable7, Enable9, Enable10;
wire [31:0] output2, output5, output8;
input clk, rst;
input [31:0] in1;
input [31:0] in2;
input [31:0] S;
output [31:0] R3;
output [31:0] R4;
output [31:0] R;
data_path dp(R3, in1, R4, in2, R, S, output2, Enable3, output5, Enable7, output8, Enable9, Enable10, clk);
control_unit cu(Enable3, Enable7, Enable9, Enable10, output2, output5, output8, clk, rst);
endmodule
```

module2.v:

```
module module2(S, output2);
input [31:0] S;
output reg [31:0] output2;
always @(S)
begin
if (S == 1) begin
output2 = 1;
end else begin
output2 = 0; end
end
endmodule
```

module5.v:

```
module module5(R1, R2, output5);
input [31:0] R1;
input [31:0] R2;
output reg [31:0] output5;
always @(R1, R2)
begin
if (R1 < R2) begin
output5 = 1;
end else begin
output5 = 0; end
end
endmodule
```

module8.v:

```
module module8(R1, output8);
input [31:0] R1;
output reg [31:0] output8;
always @(R1)
begin
if (R1 == 0) begin
output8 = 1;
end else begin
output8 = 0; end
end
endmodule
```

total_module.v:

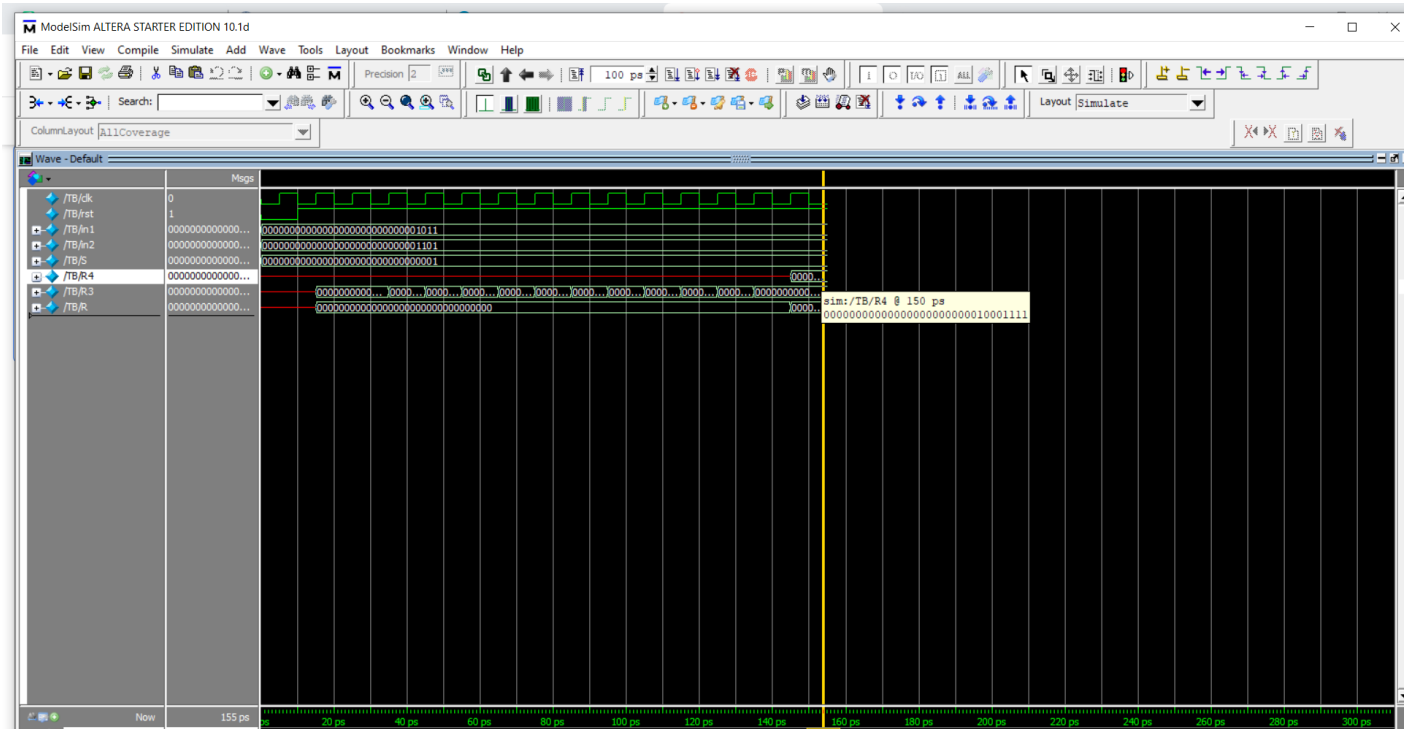
```
module total_module(Enable3, Enable7, Enable9, Enable10, R3, in1, R4, R1, in2, R, R2, clk);
output reg[31:0] R3;
output reg[31:0] R4;
output reg[31:0] R1;
output reg[31:0] R;
output reg[31:0] R2;
input [31:0] in1;
input [31:0] in2;
input Enable3;
input Enable7;
input Enable9;
input Enable10;
input clk;
always @(posedge clk) begin
if (Enable3) begin
R1 = in1;
R2 = in2;
R3 = 0;
R = 0;
end
if (Enable7) begin
R2 = R1;
R1 = R2;
end
if (Enable9) begin
R4 = R3;
R = 1;
end
if (Enable10) begin
R3 = R3 + R2;
R1 = R1 - 1;
end
end
endmodule
```

همچنین برای شبیه‌سازی کد و مطمئن شدن از عملکرد صحیح کد وریلاگ تولید یک تست بنچ به صورت دستی ساختیم و کد وریلاگ تولید شده توسط برنامه خود را به ازای یک ورودی شبیه‌سازی کردیم.

TestBench.v:

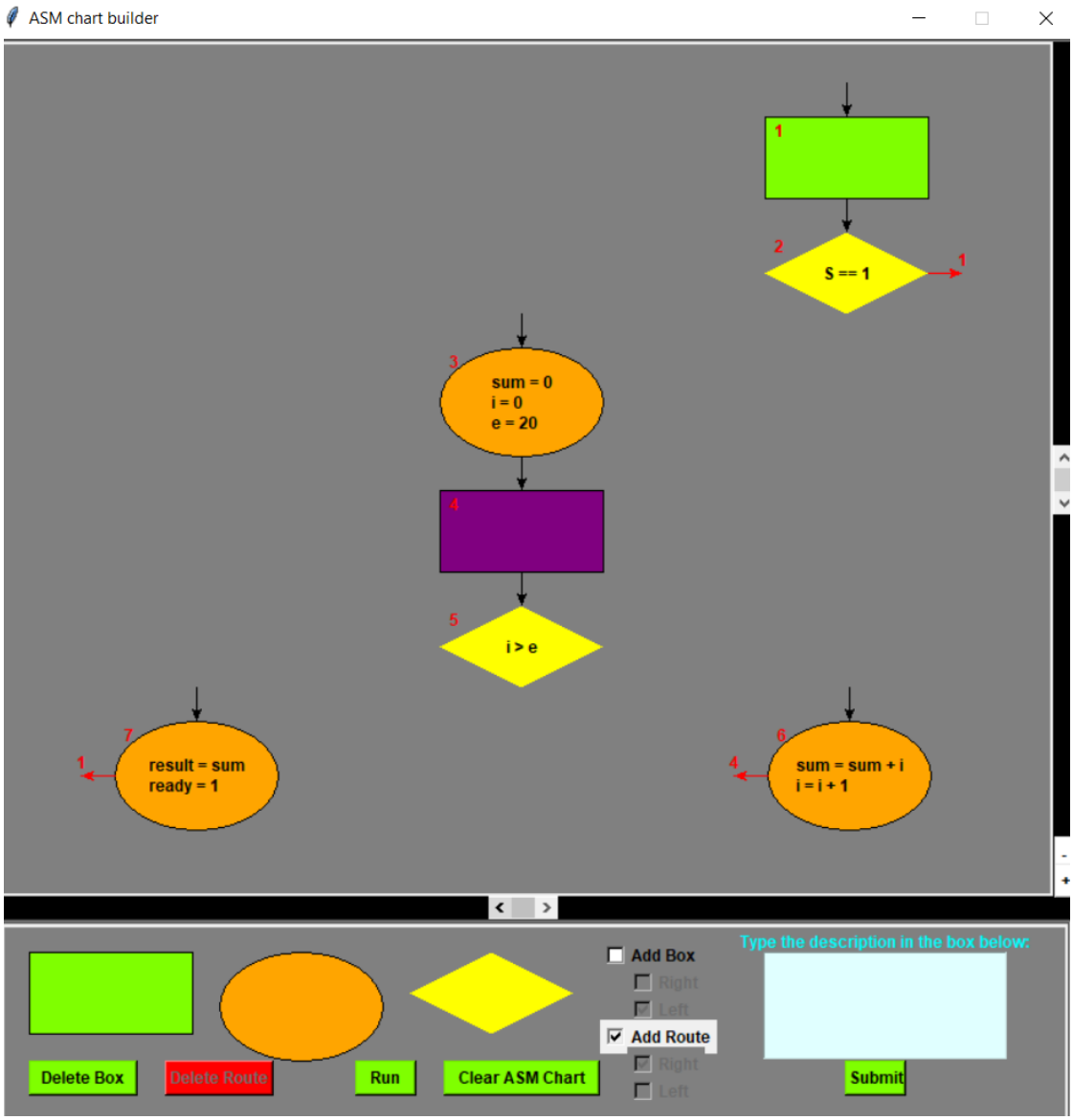
```
module TB;

reg clk, rst;
reg [31:0] in1 = 11;
reg [31:0] in2 = 13;
reg [31:0] S = 1;
wire [31:0] R4, R3, R;

main_program main_program1(in1, in2, S, R3, R4, R, clk, rst);

initial
   begin
    clk = 0;
    rst = 0;
    #10
    rst = 1;
    #145
     $stop;
    end

always
  begin
    #5
    clk = ~clk;
   end

endmodule
```

نتیجه شبیه‌سازی: ورودی‌ها: in1=11, in2=13 خروجی‌ها: R4=143, R=1



تست دوم: تست مدار جمع اعداد از ۱ تا ۲۰

نمای ASM chart در برنامه:



ماژول‌های تولید شده:

control_unit.v:

```
module control_unit(Enable3, Enable6, Enable7, output2, output5, clk, rst);
input [31:0] output2, output5;
output reg Enable3, Enable6, Enable7;
input clk, rst;
integer p_state, n_state;

always @ (posedge clk)
begin
if (rst == 1'b0) begin p_state = 1; end
else begin p_state = n_state; end
end

always @(p_state, output2, output5)
begin
begin
Enable3 = 0;
Enable6 = 0;
Enable7 = 0;
end
case(p_state)
1:
begin
if (output2 == 1) begin
begin Enable3 = 1; end
begin n_state = 4; end
end else begin
begin n_state = 1; end
end
end
4:
begin
if (output5 == 1) begin
begin Enable7 = 1; end
begin n_state = 1; end
end else begin
begin Enable6 = 1; end
begin n_state = 4; end
end
end
endcase
end
endmodule
```

data_path.v:

```
module data_path(ready, S, result, sum, output2, Enable3, output5, Enable6, Enable7, clk);
input Enable3, Enable6, Enable7;
output [31:0] output2, output5;
input clk;
input [31:0] S;
output [31:0] ready;
output [31:0] result;
output [31:0] sum;
wire [31:0] i;
wire [31:0] e;
module2 create_module2(S, output2);
module5 create_module5(i, e, output5);
total_module total_module1(Enable3, Enable6, Enable7, result, ready, sum, i, e, clk);
endmodule
```

main_program.v:

```
module main_program(S, ready, result, sum, clk, rst);
wire Enable3, Enable6, Enable7;
wire [31:0] output2, output5;
input clk, rst;
input [31:0] S;
output [31:0] ready;
output [31:0] result;
output [31:0] sum;
data_path dp(ready, S, result, sum, output2, Enable3, output5, Enable6, Enable7, clk);
control_unit cu(Enable3, Enable6, Enable7, output2, output5, clk, rst);
endmodule
```

module2.v:

```
module module2(S, output2);
input [31:0] S;
output reg [31:0] output2;
always @(S)
begin
if (S == 1) begin
output2 = 1;
end else begin
output2 = 0; end
end
endmodule
```

module5.v:

```
module module5(i, e, output5);
input [31:0] i;
input [31:0] e;
output reg [31:0] output5;
always @(i, e)
begin
if (i > e) begin
output5 = 1;
end else begin
output5 = 0; end
end
endmodule
```

total_module.v:

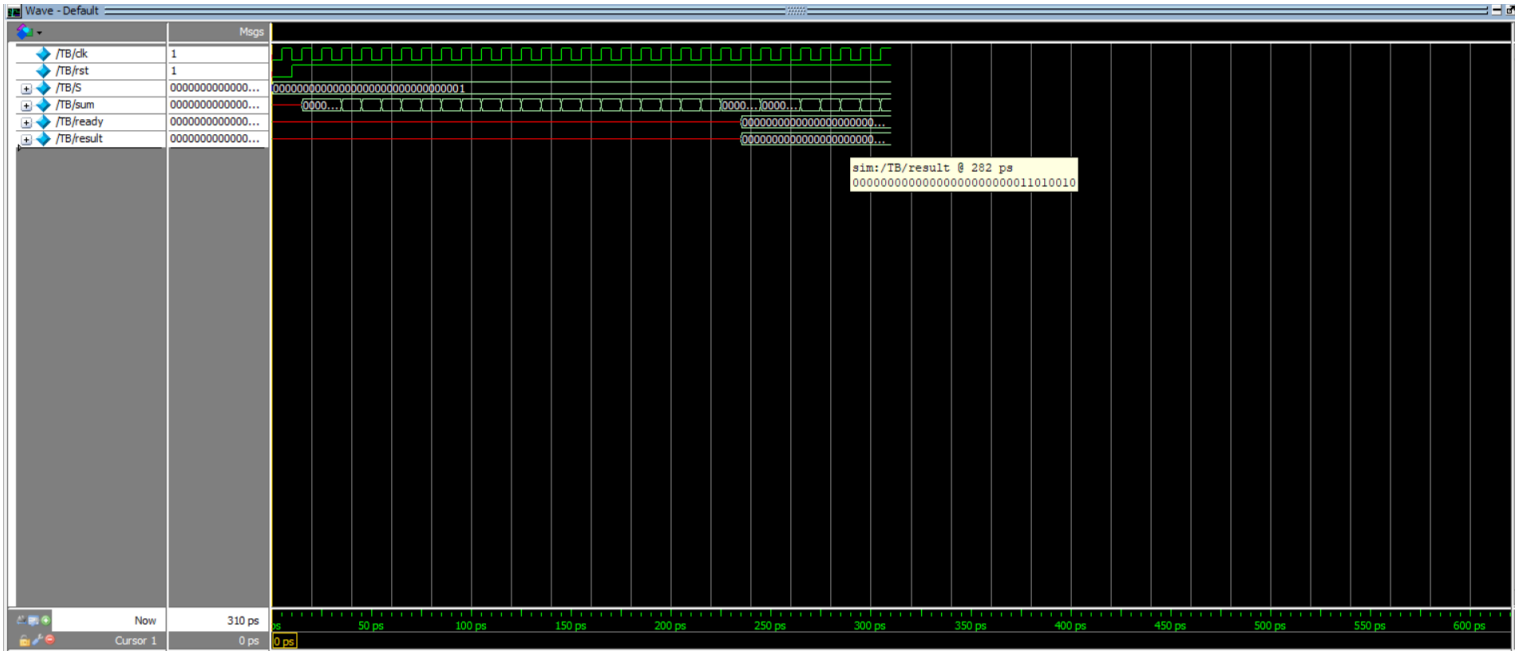
```
module total_module(Enable3, Enable6, Enable7, result, ready, sum, i, e, clk);
output reg[31:0] result;
output reg[31:0] ready;
output reg[31:0] sum;
output reg[31:0] i;
output reg[31:0] e;
input Enable3;
input Enable6;
input Enable7;
input clk;
always @(posedge clk) begin
if (Enable3) begin
sum = 0;
i = 0;
e = 20;
end
if (Enable6) begin
sum = sum + i;
i = i + 1;
end
if (Enable7) begin
result = sum;
ready = 1;
end
end
endmodule
```

همچنین برای شبیه‌سازی کد و مطمئن شدن از عملکرد صحیح کد وریلاگ تولید یک تست بنچ به صورت دستی ساختیم و کد وریلاگ تولید شده توسط برنامه خود را شبیه‌سازی کردیم.

TestBench.v:

```
module TB;

reg clk, rst;
reg [31:0] S = 1;
wire [31:0] sum;
wire [31:0] ready, result;

main_program main_program1(S, ready, result, sum, clk, rst);

initial
   begin
    clk = 0;
    rst = 0;
    #10
    rst = 1;
    #300
     $stop;
    end

always
  begin
    #5
    clk = ~clk;
  end

endmodule
```

نتیجه شبیه‌سازی: خروجی‌ها: result=210, Ready=1



با توجه به خروجی صحیح شبیه‌سازی می‌توان از عملکرد صحیح برنامه مطمئن شد.